



## Grid metacomputing support on HPCx

Stephen Booth  
[s.booth@ed.ac.uk](mailto:s.booth@ed.ac.uk)

### ***Introduction***

HPCx is part of the UK National Grid Service (NGS) and provides full support for remote job submission and file transfer using the Globus Toolkit. This is sufficient for many grid applications including those based around Job submission portals. Some Grid applications need to run in a metacomputing mode where jobs running at geographically distributed sites interact while running. This mode of operation is particularly difficult on HPCx because the backend compute nodes used for parallel jobs are only connected to private internal networks and are not able to make or receive communications from the wider Internet.

### ***File staging***

If the only requirement for network access is for file transfer stages at the beginning and end of a job then there is an easy solution. HPCx does support general Internet access for serial batch jobs. HPCx has one compute node that runs all serial batch jobs. This node also has full Internet access so file staging is easily accomplished by separating these operations off into separate serial batch jobs. Though it is possible to coordinate these stages as independent jobs, either by hand or via Globus, it is much more efficient to use the built in features of Loadleveler which allows a single batch request to consist of multiple (possibly dependant) job steps. The main advantage over using this loadleveler feature is that though loadleveler will ensure that the job steps are executed in the right order, they all enter the queues at the same time so the dependant steps are already at the front of the loadleveler queue when the jobs they are waiting for complete.

### ***Port forwarding***

In order to support metacomputing from the compute nodes we have deployed a port forwarding system. This consists of a proxy daemon running on the login node (which has full access to the Internet as well as connections to the internal networks) and a proxy library for use on the backend-nodes. Application codes are linked against the proxy library, which overrides the normal socket calls and forwards them to the proxy daemon using the standard SOCKS protocol. The daemon then makes the necessary socket connections on behalf of the application.

The port forwarding software is based on the dante package and is installed on HPCx in `/usr/local/packages/dante`. There are separate libraries for 32 and 64 bit applications.

There are two ways of enabling the port forwarder for an application. The first of these requires you to edit the C source files where the socket connections take place. By adding

```
#include "socks.h"
```

at the top of the source file. Macros are defined in this file that redirect the socket calls to the versions in the socks library. You can then compile the program adding the dante directories to your include path and linking against the socks library

```
xlc_r -I/usr/local/packages/dante/include  
-L/usr/local/packages/dante/lib.32 -o program program.c  
-lsocks
```

This method requires access to the source code where the socket operations takes place and is therefore inconvenient when the socket operations take place in a third party library such as Globus. We therefore provide an alternative mechanism that only requires the program to be re-linked. The dante library directories include a **socks\_wrapper.o** object file. If you include this object file at the link stage all socket calls in your application will be redirected to the socks daemon.

```
xlc_r -L/usr/local/packages/dante/lib.32 -o program  
/usr/local/packages/dante/lib.32/socks_wrapper.o  
program.o -lsocks
```

It is even possible to re-link an existing executable in this way provided that the executable has not been stripped.

Note that the socks library will not work correctly if the executable is running on the same computer as the proxy daemon. For this reason if you need to run the same program on the login node as well as the backend nodes then you will need to have two version of the program, one linked against the socks library and one linked normally.

## **MPICH-G2**

One of the standard ways of implementing metacomputing applications is via the MPICH-G2 library. This is a special version of MPI that is built on top of Globus and the native MPI libraries of a system (IBM MPI in our case).

MPICH-G2 allows the application author to write a metacomputing application using the normal MPI calls. Wherever a communication has to take place within a single system/cluster the communication takes place using the native MPI library. Whenever a communication takes place between systems the Globus-IO library (based on sockets) is used. On HPCx you need to use the socket forwarder as well as MPICH-G2. For example:

```
/usr/local/packages/globus/mpich-1.2.7_32/bin/mpicc  
-L/usr/local/packages/dante/lib.32 -o hello  
/usr/local/packages/dante/lib.32/socks_wrapper.o hello.c
```

MPICH-G2 jobs are submitted via Globus using **globusrun** and a RSL script for example:

```

+
( &(resourceManagerContact="login.hpcx.ac.uk/jobmanager-
loadleveler")
  (count=2)
  (jobtype=mpi)
  (label="subjob 0")
  (environment=(GLOBUS_DUROC_SUBJOB_INDEX 0)
    (GLOBUS_TCP_PORT_RANGE 65200,65255)
    (GLOBUS_HOSTNAME login.hpcx.ac.uk)
    (MP_LABELIO YES)
  )
  (directory="/hpcx/home/z001/z001/spb/Globus")
  (executable="/hpcx/home/z001/z001/spb/Globus/sring")
  (project="z001")
  (stdout="/hpcx/home/z001/z001/spb/Globus/ring.out")
  (stderr="/hpcx/home/z001/z001/spb/Globus/ring.err")
  (max_wall_time=1)
)
( &(resourceManagerContact="lomond.epcc.ed.ac.uk/jobmanag
er-sge")
  (count=2)
  (jobtype=mpi)
  (label="subjob 1")
  (environment=(GLOBUS_DUROC_SUBJOB_INDEX 1)
    (GLOBUS_TCP_PORT_RANGE 65200,65255)
  )
  (directory="/home/spb")
  (executable="/home/spb/ring")
  (stdout="/home/spb/ring.out")
  (stderr="/home/spb/ring.err")
  (max_wall_time=1)
)

```

Note that we explicitly set the GLOBUS\_HOSTNAME to be the HPCx login node. Though the actual jobs will run on the backend nodes all socket connections will appear to be made from the login node so to the rest of the Globus application it appears that all jobs are running on the login node and this setting ensures that the HPCx processes report this as their hostname.

It is also important to ensure that all the different executables and the globusrun program used to launch the job are all compiled with the same word-size flavour (64-bit or 32-bit).

### ***Job co-scheduling***

Another important requirement for metacomputing is the ability to co-schedule jobs on different machines so that the different parts of the metacomputing job start to run at approximately the same time. Without this whichever job starts first will probably come to the end of its allocated run-time long before the other parts of the job have even started. This is less of an issue on lightly loaded test systems where jobs can be expected to start almost immediately but on HPCx queue wait times of many hours are quite common. We plan to eventually support co-scheduling on HPCx via the reservation mechanism in loadleveler. Loadleveler reservations allow a user to reserve some number of nodes for a specified time period and batch jobs can be tied to a reservation so they run on the reserved nodes. Reservations need to be made well in advance and as the start time of the reservation approaches loadleveler will stop allocating jobs to the reserved nodes in order to ensure that the nodes are available at the specified time. Inevitably this will result in nodes sitting idle as the reservation time approaches so node reservations will probably be charged at a premium rate.

### ***Metacomputing performance***

Though MPICH-G2 allows applications to be written using the normal MPI interface, efficient metacomputing is fundamentally difficult to achieve and normal HPC application codes will not run well as meta-computing applications. This is due to radically different performance characteristics of HPC interconnects and long haul networks. Consider the following basic communication metrics.

**Single message latency** - The minimum time taken to send a message (zero length message time).

**Single message bandwidth** – The maximum rate at which data for a single message can be transferred (speed of large messages)

**Application bisection bandwidth** – The limit on the rate that one half of an application can exchange data with the other half.

Within HPCx the MPI message latency is approximately **6 microseconds**. The single message bandwidth is approximately **3 GByte/s** this is mostly limited by the speed of the network adapters so each Lpar (16 processors) is limited to an aggregate bandwidth of approximately 3.5 GByte/s simultaneously in each direction. However the switch network scales well with the number of Lpars so the Application bisection bandwidth of a 512-processor job running on HPCx is **50 GByte/s** simultaneously in each direction.

If we compare this to a MPICH-G2 job running between HPCx and a machine in Edinburgh. The measures MPICH-G2 latency is 4 milliseconds (**~4000 microseconds**). This is consistent with half the round-trip latency as reported by ping and traceroute, which

suggests that these tools may be used to predict the MPICH-G2 latency to other remote sites.

The measured single message bandwidth was only **4MByte/s**. The limiting factor in this case was almost certainly the final network section at Edinburgh which only runs at 100Mbit/s (peak speed =  $100/8 = 12.5$  MByte/s). However the external network interfaces for the HPCx login node are rated at 1000Mbit/s so the maximum single message and bisection bandwidths achievable for metacomputing application on HPCx are 125MByte/s = **0.125 GByte/s**. Taken together this means that to be effective metacomputing applications need to be written to be very tolerant of high message latencies and be decomposed in such a way as to tolerate very restrictive bandwidths between the sections of the calculation assigned to different remote sites.

It is also interesting to predict how future advances in technology may affect these numbers. There is absolutely no prospect of any significant improvement in latency. The distance between HPCx and Edinburgh is roughly 300Km. The speed of light in optical fibre is roughly 2/3 of the speed of light in a vacuum so the laws of physics prevent this latency from being anything less than 1.5 milliseconds. However currently the restricted bandwidth is probably more detrimental to application performance. The current state of the art for networking technology (as illustrated by the soon to be deployed SuperJanet-5 network) is 10 Gbit/s = 1.25Gbyte/s links. It would require 40 10Gb/s links in parallel to support a bisection bandwidth matching that currently enjoyed by applications running on HPCx. However modern networking equipment like that used in the Superjanet-5 backbone can support multiple parallel channels through a single piece of optical fibre using different light wavelengths so the bandwidth limitation is potentially solvable.

We can therefore deduce that converting a normal HPC application into one that can run on a geographically distributed meta-computer is potentially a very difficult task and may even be impractical for some types of application. However the Grid community has developed a number of successful metacomputing applications such as CACTUS and NAMD. These applications reduce the impact of the network latency by overlapping communication and calculation and by restructuring the communication pattern of the application to require less frequent but larger messages.